Architectural Terminology

Software Architecture

- A structured solution that meets all of the technical and operational requirements for a given piece of software
- Optimizes common quality benchmarks
  - performance
  - security
  - ease of maintenance
  - scalability

Framework

- A conceptual body of interoperable code that may be utilized as a subset of functionality within a larger architectural solution
- Contains generic functionality which can be overridden and customized by developers for specific use cases
- Commonly referred to as libraries.

Domain

- Conceptual problem space that an architectural solution is created to address

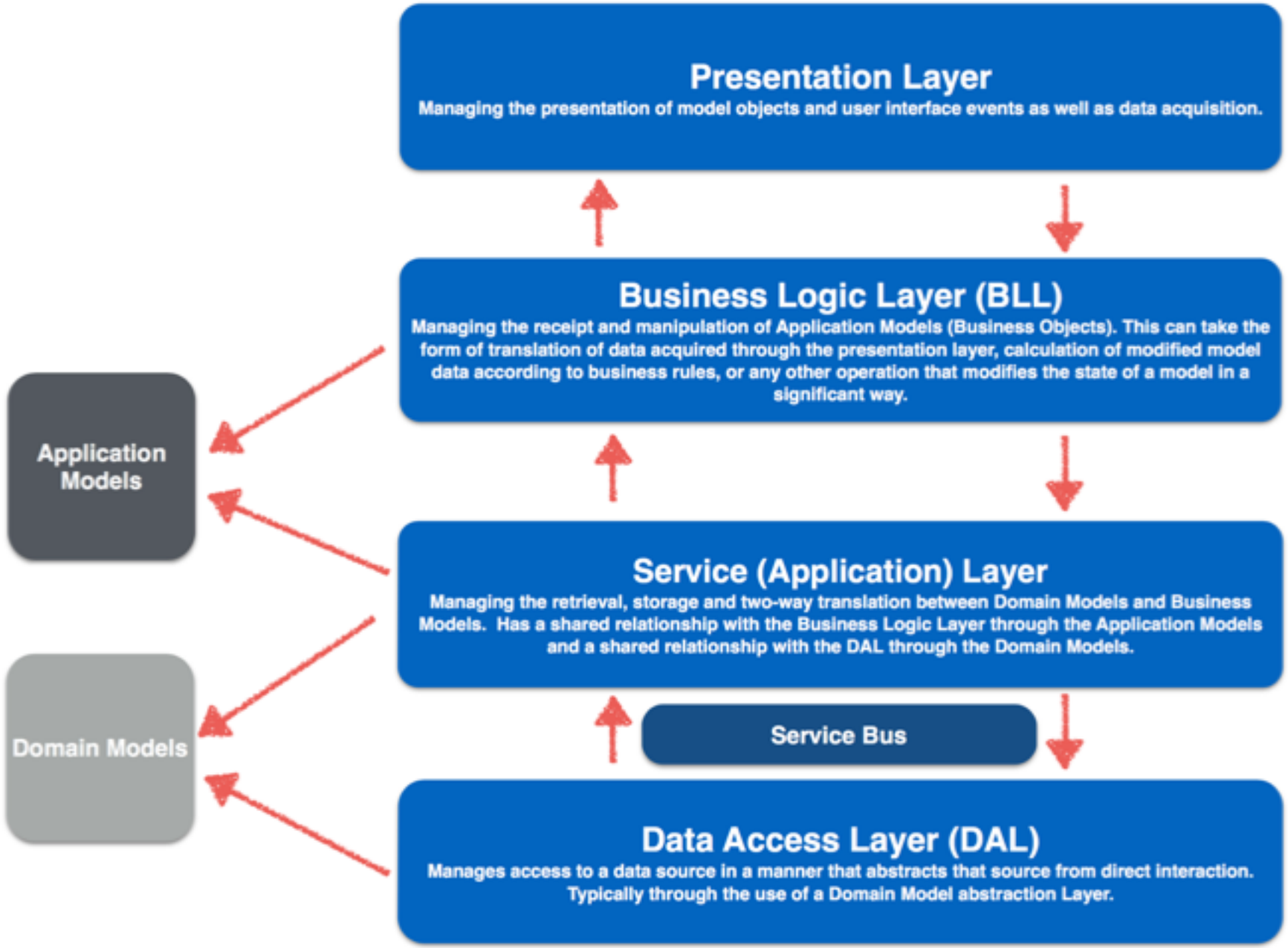
Model

- Object representing a conceptual data component

- Domain Model
  - A model in the domain space
  - Solution-independent
- Application Model
  - A model in the space of the application solution in the domain
  - May be different from domain models due to requirements of the solution
    - For example, may have different models based on the platform

N-Tier/Multi-Tier Architecture

- Client-to-server architecture where presentation, business logic, services, data management are logically divided into abstraction layers to support separation of concerns


Presentation Layer
Managing the presentation of model objects and user interface events as well as data acquisition.
Business Logic Layer (BLL)
Managing the receipt and manipulation of Application Models (Business Objects). This can take the form of translation of data acquired through the presentation layer, calculation of modified model data according to business rules, or any other operation that modifies the state of a model in a significant way.
Application Models
Service (Application) Layer
Managing the retrieval, storage and two-way translation between Domain Models and Business Models. Has a shared relationship with the Business Logic Layer through the Application Models and a shared relationship with the DAL through the Domain Models.
Domain Models
Service Bus
Data Access Layer (DAL)
Manages access to a data source in a manner that abstracts that source from direct interaction. Typically through the use of a Domain Model abstraction Layer.

Abstraction Layer

- A logical means of hiding implementation details for a particular set of functional concerns.
- Allows a looser coupling between functional concerns and naturally enforces the need for reusable components (supports DRY).
- Also allows independent testing of different functional areas within an application.

Data Source / Data Storage

- Repository for domain data
- Typically represented by logical entities in relational schema
  - relational databases

Data Access Layer (DAL)

- An interface that simplifies access to data storage
- Uses definition of domain models, derived from domain specific schema
- Shares relationship with Application Layer through the Domain Models

Data Transfer Object

- Dumb object (remember OOP)
- No logic beyond what is necessary to provide read/write access
- Typically describes model objects that are serialized for transmission over a service bus

Service Bus

- Technology in place for communicating data to and from the DAL
- Typically transmits in a serialized format

Service/Application Layer

- Manages the retrieval, storage and two-way translation between Domain Objects and Business Objects.
- Shares relationship with the DAL through the Domain Models.
- Shares relationship with the Business Logic Layer through the Application Models

Business Logic Layer

- Manages receipt and manipulation of Application Models/Business Objects
- Can be several forms:
  - Translation of data from Presentation Layer
  - Calculation of modified model data according to business rules
  - Any other operation that modifies the state of a model
- Shares relationship with Application Layer through the Application Models

Presentation Layer

- Manages presentation of data and user input
- Displays the data it receives from the Business Logic Layer
- Passes information on what the user does back to the BLL
  - The Presentation Layer does not modify any data itself!

Serialization/Marshalling Deserialization/Unmarshalling

- The process of converting a DTO to/from a stream of bytes for transmission via a service bus
  - Can also be serialized for storage
- Does not include methods for the object
- Does support complex variable types (not just primitive types)

Abstract Terminology

User Story

- Description of a feature from point of view of intended user base
- Provides a real-world use case for a feature
  - this is different from a software requirement
- Multiple user stories may be necessary to describe a single feature requirement

Software Feature

- Any property of a software application notable for its unique function or use within the application space

Feature Vertical

- The multi-tier implementation of a given feature
- All the way from Presentation to Data

Platform (Computing)

- Manufactured environment that a particular piece of software is designed to run in
- Typically refers to a particular operating system and/or the device it runs on
  - e.g. iOS and mobile Apple devices are practically synonymous

Platform (Software)

- Software that provides a platform for building other software
- Differs from Frameworks
  - A Platform can run independently
  - A Framework requires placement within a given runtime environment and usage of Framework elements to run

Software Design Principles

Separation of Concerns

- Division of a software application into several features
  - Individually distinct
  - Minimal amount of overlapping functionality
- Must ensure separation occurs at appropriate boundary
  - Otherwise risk increased complexity and tighter coupling

Single Responsibility Principal

- Each modular piece of a software application is responsible for a single, specific feature or level of functionality
- Exception to rule: when responsibility of module or component is the composition of multiple sub-modules

Principle of Least Knowledge

- Any given module, component, or object should have no knowledge of the internal workings of any other module, component, or object

Don’t Repeat Yourself (DRY)

- Functional code implemented in a given module or component should be implemented in that component only and not duplicated in any other component

Minimize Design at All Stages

- At all stages of development, only undertake the minimum design necessary to realize the Minimum Viable Product (MVP)
- Avoid designing for unclear or unspecified requirements
  - called ”over-architecting”
- Avoid designing for “what-if” scenarios